What do we know about the defect types detected in conceptual models?

Granda, Maria Fernanda; Condori-Fernandez, Nelly; Vos, Tanja E.J.; Pastor, Oscar

published in
2015 IEEE 9th International Conference on Research Challenges in Information Science (RCIS)
2015

DOI (link to publisher)
10.1109/RCIS.2015.7128867

document version
Publisher's PDF, also known as Version of record

What do we know about the Defect Types detected in Conceptual Models?

Maria Fernanda Granda*
Department of Computer Science
University of Cuenca
Cuenca, Ecuador
fernanda.granda@ucuenca.edu.ec

Nelly Condori-Fernández
Department of Computer Science
VU University Amsterdam
Amsterdam, Netherlands
n.condori-fernandez@vu.nl

Tanja E.J. Vos, Oscar Pastor
*PROS Research Center
Universitat Politècnica de València
Valencia, Spain
{tvos, opastor}@pros.upv.es

***Abstract*—In Model-Driven Development (MDD), defects are managed at the level of conceptual models because the other artefacts are generated from them, such as more refined models, test cases and code. Although some studies have reported on defect types at model level, there still does not exist a clear and complete overview of the defect types that occur at the abstraction level. This paper presents a systematic mapping study to identify the model defect types reported in the literature and determine how they have been detected. Among the 282 articles published in software engineering area, 28 articles were selected for analysis. A total of 226 defects were identified, classified and their results analysed. For this, an appropriate defect classification scheme was built based on appropriate dimensions for models in an MDD context.**

Keywords— Conceptual Schema Defects, Defect Classification Scheme, Systematic Mapping Study, Model-Driven Development

I. INTRODUCTION

Unlike in traditional Software Development where the software is the main artefact, in Model-Driven Development (MDD) the main artefact is a model (conceptual schema -CS[1]). Instead of building and rebuilding software through programming, during MDD the Conceptual Schemas are extended and transformed into other models to build the software product using code-generation strategies.

CSs are developed using a modelling language. The de-facto standard for analysis and design of object-oriented software systems is the Unified Modelling Language (UML) [1] [2], which is extended with OCL (Object Constraint Language) constraints [3]. The variety of UML diagrams provide flexibility and applicability to modellers to create CSs in the different spaces where they can be used (problem, solution and background) [4]. However, since the modelling process is a human task, it is difficult to avoid introducing defects into the CSs (e.g. inconsistency, incorrect, redundant and imprecise elements).

Although defects may be inevitable, we should minimize their number and impact on software quality through testing and/or inspecting the CS. Information on the defect types that occur in the earlier stages of the software development life cycle can be used to give feedback to stakeholders (e.g. modellers, developers, testers) about detecting defects and how they can be tracked, reduced and resolved. Moreover, if the purpose is to get a good quality CS, the information on each defect must be related to the quality goals affected, according to an appropriate quality model for models in a MDD context, as proposed in [5].

Defect classification schemes define a set of attributes[2], attribute values and a process to classify the defects. Freimut [6] claims that "*Defect classification schemes can differ in the way different attributes or attribute values relate to each other"*. For example, for indicating the location of a defect in code, a program line number will suffice. For a UML model, the defect is recognized by referring to the element (i.e. class name) and the diagram name. So far, existing defect classification schemes (e.g. HP scheme [7], Orthogonal Defect Classification (ODC) by Chillarege [8] and IEEE std. 1044 [9]) are not appropriate for CSs because they were designed for other contexts (e.g. for defects at implementation and code level), giving rise to the need for adaptations [10], [9]. Few attempts can be found in the literature to classify defects in CSs [11], [12], [13], [14], [15], [PS1][3] and [PS21]. However, none of these uses a defect classification scheme to systematically capture the defect information (e.g. description, affected quality goal, technique type) in order to classify the defects and identify the group to which it belongs.

The objective of the study described in this paper is to get an overview of the type of defects that are reported in the literature at the CS level and determine how and where they have been detected. Therefore, we first build a classification scheme for CS defects based on standard classification schemes to extract clear and complete information on defect types that occur in CSs. Subsequently, we executed a mapping study to obtain an overview and categorize the information published in relation to defect classifications and/or taxonomies. In this paper, we present the results of our systematic mapping study based on guidelines proposed by Kitchenham et al. [16] and Petersen et al. [17]. This work is part of a more extensive research study, whose main goal is to develop an approach for testing-based

[1] A conceptual schema is a description, representation or definition of the knowledge that is needed by an information system (requirements) to perform its functions [*25*]. In this work the terms "conceptual schema", "conceptual model" and "model" are considered similar.

[2] In this paper, the term attribute is used to refer to a feature or property of a defect.

[3] References beginning with ''P" refer to primary studies covered in this mapping study as given in Appendix II.

978-1-4673-6630-4/15/$31.00 ©2015 IEEE

conceptual schema validation in a MDD context [18], [19]. This work particularly focuses on CSs in UML-based systems.

The remaining parts of this paper are organized as follows: Section 2 presents the background on CS quality models. Section 3 summarizes the process of building a classification scheme for CS defects. Section 4 describes the method used in conducting the mapping study. Section 5 discusses the results of this review and defect classification. Section 6 presents the limitations of this study. Finally, Section 7 presents the conclusions and future work.

II. Background: Quality Models for Conceptual Schemas

Different quality models can be found in the literature for describing the quality of CSs (e.g. [20], [21], [4] and [22]). In the present paper we have chosen Mohagheghi et al. [5] as a reference because they describe a quality model oriented to Model Driven Engineering (MDE). Their model takes into account that an MDD approach automates many activities in software development, and consequently CSs in MDD are expected to get progressively more complete, precise and executable such that in the end they can be used to generate the code and other artefacts such as test cases. MDD therefore adds new requirements to the development process such as consistency between models, technical comprehension by tools and support changes. They perform a combination of several quality models and identify the following six classes (6C) of CS quality goals, as follows (see Table I).

TABLE I. Quality Goals (based on 6C quality model from Mohagheghi et al. [5])

Quality Goal (QG)	Description
Correctness QG1	Including correct elements and relations between them, and including correct statements about the domain; not violating rules and conventions; for example adhering to language syntax. Thus it covers both syntactic correctness (right syntax or well-formedness) and semantic correctness (right meaning and relations relative to the knowledge about the domain).
Completeness QG2	Having all the necessary information that is relevant and being detailed enough according to the purpose of modelling. It is a semantic quality.
Consistency QG3	Having no contradictions in the models, related to syntactic quality. It covers consistency between views that belong to the same level of abstraction or development phase (horizontal consistency), and between views that model the same aspect, but at different levels of abstraction or in different development phases (vertical consistency). It also covers semantic consistency between models; i.e, the same element does not have multiple meanings in different diagrams or models.
Comprehensibility QG4	Being understandable by the intended users, either human users or tools. It is related with the pragmatic quality.
Confinement QG5	Being in agreement with the purpose of modelling and the type of system, and being restricted to the modelling goals; such as including relevant diagrams and being at the right abstraction level. It is related with the semantic quality.
Changeability QG6	Supporting changes or improvements so that models can be changed or revolved rapidly and continuously. It is related with the pragmatic quality.

Our paper considers the aforementioned quality goals and relations between them to classify CS defects and analyse the results.

III. Classification Scheme For Conceptual Schema Defects

In order to address the goal of our work we first needed to define a defect classification scheme (DCS) previously to data extraction of the literature. This process was conducted in the following three stages:

A. Establish a list of Defect Causes

First, we needed to establish a list of defect causes that enables the classification and documentation of a found defect.

We decided to define the defect causes at a coarse-grained level by taking the modes from the IEEE standard [9], i.e. missing, wrong and unnecessary. Then, in order to relate the cause of the defects with affected quality goals proposed by Mohagheghi et al. [5], we specified sub modes (types) of the defects related to the quality goals affected by them (see Table II). These sub modes are based on definitions from [9], [7], [23], [11], [PS8] and [PS26], and have been adapted to CSs.

TABLE II. Defect types by mode and their relation with the Quality Goals

Mode (defect cause)	Sub modes	Affected QG
MISSING Something is absent that should be present [9].	Missing	QG2, QG4
WRONG Something is incorrect, inconsistent or ambiguous [9].	Inconsistent: There are contradictions in the models (vertical and horizontal inconsistency) [9], [PS8], [PS26].	QG1, QG3, QG4, QG5
	Incorrect: There is a misrepresentation of concepts about the domain, their attributes and their relationships, as well as the violation of the rules by combining of these concepts at the time of building partial or complete models [9], [7], [11], [23].	QG1, QG4
	Ambiguous (wrong wording): The representation of a concept in the model is unclear, and could cause a user (modeller, low-level designer, etc.) to misinterpret or misunderstand the meaning of the concept [9], [23], [11].	QG1, QG3
UNNECESSARY (Extra) Something is present that need not be [9].	Redundant: If an element has the same meaning that other element in the model.	QG5
	Extraneous: If there are items that should not be included in the model because they belong to another level of abstraction (e.g. details of implementation) [23], [11].	QG5, QG6

B. Construct a Defect Classification Scheme

Following a slight adaptation of the process proposed by Freimut [6], we started by identifying those attributes that were considered important to register and classify, once a defect was found in a CS. We used terminology aligned with UML concepts [2] (modelling language used for building the CSs) and based on the standard IEEE classification [9] and its guide [24].

First, we adjusted the description of the scope of the relationships between conceptual entities proposed by the standard IEEE on the one side, with the conceptual entities of our study (UML-based conceptual models) on the other. This resulted in Fig. 1, where these relationships are depicted graphically. The red frame directly corresponds to the IEEE standard.

Looking at Fig. 1, a conceptual schema represents the (software) systems requirements at an abstract level. It may consist of several UML diagrams (structural and behavioural), where each diagram type contains different type (modelling element) of information about the system. Additionally, the conceptual schema has associated quality properties that support the representation or description of the requirements. These quality properties usually are threatened by defects that occur at the diagrams elements level of the conceptual schema.

A defect may be associated with a single Corrective Change Request of the Conceptual Schema, which attempts to resolve the defect and each Corrective Change Request may be associated with, at the most, a single Conceptual Schema Release. The Fig. 1 also represents the other two causes of Conceptual Schema Changed Request (CSCR), perfective change request of conceptual schema and adaptive change request of conceptual schema.

Additionally, the defects at conceptual level can be located in several ways through V&V techniques, which use a detection mechanism (based on rules, metrics, and modelling conventions) for this purpose. According to the technique nature, this can be static or dynamic supported by a tool and they can have different scope depended on its purpose (i.e. detect, prevent and resolve).

Finally, the defects have insertion activity, severity, priority and probability of occurrence. They are detected at any specific time by noticing specific description (symptom), using detection mechanism. Each of these aspects is relevant for the purpose of required analysis and also allows a classification of defects.

Based on this analysis, we took some attributes from the standard IEEE and adapted them to the context of UML-based models, then added attributes required in this study (i.e. technique purpose, technique type and tool support and detection mechanism). Table XI (see Appendix I) documents the mapping between attributes of the proposed classification scheme and provided by the standard IEEE. Once we finished this mapping, we could document the attributes and attributes values.

The set of values for each attribute was obtained from definitions of Table II (i.e. defect cause, sub mode), UML definitions (i.e. modelling element, diagram level and diagram type), standard IEEE (i.e. priority) and defects found in the literature (i.e. detection mechanism, tool support and references).

The final classification scheme was compiled and depicted in the Table III with the description and attribute values to each scheme attribute.

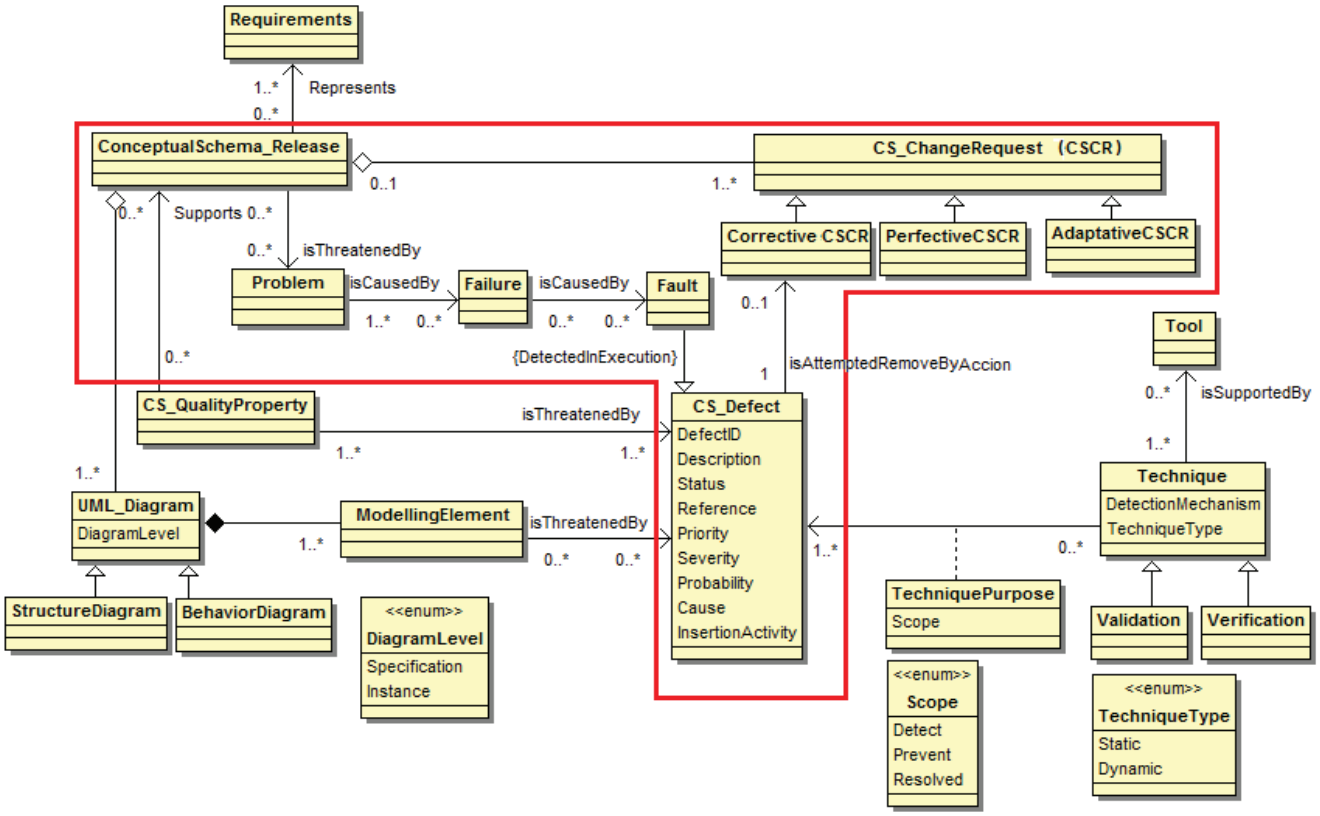

Fig. 1. Relationships modeled for understanding the relationships among conceptual entities used in Classification Scheme Structure (adapted from [9])

TABLE III. ATTRIBUTES OF DEFECT CLASSIFICATION SCHEME

Attribute	Description	Attribute Values
Defect Cause (Mode)	A categorization based on Table II.	*Missing, wrong or unnecessary* (see Table II).
Sub mode	A categorization at a fine-grained level of the defect causes.	Missing, inconsistent, incorrect, ambiguous, redundant or extraneous (see Table II).
Defect ID	Unique identifier for the defect.	*DX.Y*, X=sub mode (1-6), Y=sequential number that identifies the defect within the sub mode.
Description	The term or phrase used to describe the defect of the CSs.	Short *phrase* using words according to defect causes (see Table II)
Modelling Element	The design entities involved in the defect.	*Element* of UML diagrams (e.g. classes, relationships, operations, objects, messages).
Diagram Level	These levels are according the UML diagrams [*2*].	***Specification***= This level contains elements that represent specifications for instances, such as classes, associations and messages. ***Instance***= The instance level contains instances for diagrams elements such as objects and links.
Diagram Type	It is a categorization based on the diagram within which the defect is found. A distinction between behaviour diagrams and structure diagrams is performed.	e.g. Use Case (*UC*), class diagram (*CD*), object diagram (*OD*), sequence diagram (*SD*), communication diagram (*CoD*), activity diagram (*AD*), state machine diagram (*STMD*) and package diagram (*PD*).
Priority	Describes the defect priority.	***High*** (H), ***Medium*** (M) and ***Low*** (L).
Severity	Describes the severity of a resulting or potential failure with respect to CS. Various levels of severity can be found in the primary studies. A translation to the following terms is required.	***Critical*** = a defect that will cause the model execution to crash, stop or close abruptly. ***Medium*** = a defect, which will cause an observable failure or breach of requirements. ***Low*** = a defect that will not cause a failure in execution of the CS.
Technique Type	Whether to detect the defect execution of the CS is required or not.	***Static*** (S) = execution is not required. ***Dynamic*** (D) = execution is required.
Detection Mechanism	Detection mechanism name used by the detection technique.	The word o words used to name the detection mechanism.
Tool Support	Tool name used for detecting process.	The word o words used to name the tool support.
References	List of references	Source ID =[PSn]

C. Definition of the Defect Classification Process

To classify the defects found in the selected literature, we made a list of questions based on the defect classification process adapted from IEEE [*24*]. This process basically consists of three steps: (a) identifying the attribute information, (b) classifying the attribute information, and (c) recording the attribute values of the defect. Our process basically defines a user guide with questions previously formulated for the person recording the defect and that can be conducted along the classification process by answering each question and at the end get the possible location of the defect within the structure of the scheme classification (see Table IV). The attributes defect identifier (Defect ID) and mode (defect cause) are assigned later based on the sub mode value of the defect.

TABLE IV. QUESTIONS FOR CLASSIFYING THE DEFECT

Steps of Defect Classification Process	Attribute Name	Question
Defect Recognition	Sub mode	What is missing, inconsistent, incorrect, ambiguous, redundant, or extraneous?
	Description	How did the defect manifest itself? (e.g. missing class)
	Modelling Element	Which diagram element contains the defect? (e.g. class, association, message)
	Diagram Level	What does level of the diagram is affected? (specification or instance)
	Diagram Type	Which diagram contained the defect? (e.g. CD, SD)
	References	Where (paper) was reported the defect? (e.g. PS8)
Impact Identification	Priority	What is the importance of resolving the defect?
	Severity	How severe is the defect with respect to quality of conceptual schema? (e.g. high, medium)
Detection Investigation	Technique Type	Which type of technique can detect it? (e.g. static)
	Detection Mechanism	Which is the detection mechanism used by the technique? (e.g. automated inspection, checking consistency rule)
	Tool Support	What does tool can detect/resolve/prevent it? (i.e. tool name)

IV. DESIGN OF THE MAPPING STUDY

The main steps in conducting a mapping study are discussed in the following subsections.

A. Defining the Research Questions

As mentioned in Section 1, this paper focuses on summarizing and classifying the CS defects reported in the literature. The CSs we consider are focused on information systems design based on UML. Therefore, this work answers the following research questions (RQs):

- RQ1: What defects in UML-based CSs are reported in the literature?
- RQ2: How and where have these defects been detected?

B. Search Strategy for Selection of Primary Studies

A Systematic mapping was conducted for a period of three months (October to December/2014). SCOPUS™[4] was used as the search engine, as it provides access to well-known bibliographic-databases such as IEEExplore, SpringerLinks and

[4] www.scopus.com

ACM Digital Library, etc. A search string was created based on keywords extracted from the research questions and augmented with synonyms. After iterative refinement, the definitive search string was:

SS: (defects OR faults) AND (UML model OR UML design OR UML diagram)

Executing the search string in SCOPUS™ on article title, abstract and keywords the result was an extensive list of candidate papers for the review.

We restricted these preliminary search results by limiting the subject area to computer science and the document type to conference paper, article and book chapter in English. To ensure that only relevant studies were included, sets of inclusion and exclusion criteria were defined (see Table V).

TABLE V. INCLUSION AND EXCLUSION CRITERIA

Inclusion criteria	Exclusion Criteria
I1. Papers about defects or faults in CSs based on UML in particular and how or where defects have been detected.	E1. Papers that do not comply with the inclusion criteria presented.
I2. Studies available online.	E2. Informal literature e.g. editorials, keynotes, introductions to/abstract, posters and slides alone.
I3. Studies written in English.	E3. Duplicated reports (the most complete version of the work was included in the review).

A two-phase selection process was performed as follows:

1) In the first search phase, based on the titles, keywords and abstracts, irrelevant papers were excluded. A total of 28 papers were identified in this phase.

2) In the second search phase, we read the remaining papers and eliminated any that were not related to the research question and identified 11 papers. However, in this phase 53 additional studies were located by scanning the references and grey literature (e.g. technical reports and theses) for any other candidate primary sources. The two selection phases were again applied and a further 17 papers were added the list.

We identified a total of 28 primary studies (see Table VI).

TABLE VI. SELECTED PAPERS IN EACH PHASE OF SYSTEMATIC MAPPING

Search	**# of hits**	**Phase 1 (titles, keywords and abstracts)**	**Phase 2 (full text)**	**Papers selected**
SS	282	28	11	[PS2] [PS3] [PS4] [PS15] [PS16] [PS21] [PS22] [PS24] [PS25] [PS27] [PS28]
Additional from References		53	17	[PS1] [PS5] [PS6] [PS7] [PS8] [PS9] [PS10] [PS11] [PS12] [PS13] [PS14] [PS17] [PS18] [PS19] [PS20] [PS23] [PS26]
Total	282	81	28	

Table XII, provides a list the selected studies numbered from PS1 to PS28. This table is given in Appendix II. These results include 5 journal papers, 3 workshops, 19 conference papers and 1 Ph. D thesis. There are 4 primary studies on industrial evaluation, i.e. [PS15], [PS16], [PS17] and [PS18], the others were generated in the academic area.

Fig. 2 shows the yearly distribution of primary studies found in the literature, where 2006 is the year with more papers reporting CS defects.

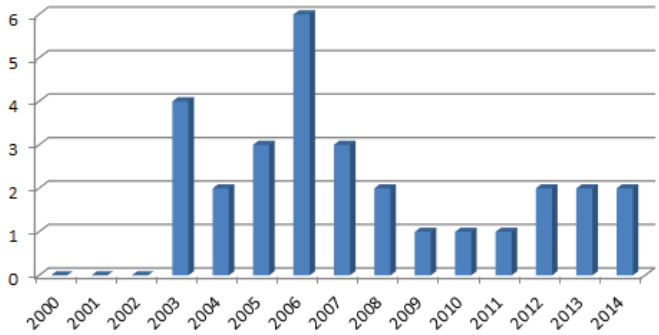

Fig. 2. Papers reported in the literature by years.

C. Data Extraction

In this step we used the proposed classification scheme (see section III) to classify the CS defects reported in the literature.

In this way 226 reported defects were identified and classified in 100 different defects.

All the rows of Table III were stored on a spreadsheet, which was used as the prototype tool for storing and analysing defect data.

An overview of this defect list is given in Table XIII (see Appendix III) in which some attributes (i.e. sub mode, diagram level, severity, detection mechanism and tool support) were omitted due to space restrictions.

The complete list of defects can be accessed through the URL http://users.dsic.upv.es/~nelly/defects.htm.

V. REVIEW RESULTS

This section deals with the results of the systematic mapping study based on the 28 papers selected. The results are structured based on the Research Questions given in Section IV.

A. RQ1: What defects in UML-based CSs are reported in the literature?

By using the attributes of our defect classification scheme (e.g. defect cause, modelling element), several groups of defects were identified. In order to answer this question, we counted the number of papers that reported each defect type. The frequency by defect type is given in Table VII.

From these results we can see that the most commonly reported defect is the "Wrong" type, with 182 defects (81%), the

most frequently reported sub-types (modes) are: Incorrect (95 defects, 42%) and Inconsistent (75 defects, 33%). Extraneous is the least frequently reported (6 defects). The least reported defect type is "Missing", with 18 defects (8%). In the "Unnecessary" defect type, the sub mode Extraneous is the least reported (6 defects, 3%) in comparison to the redundant sub-type (mode).

TABLE VII. NUMBER OF DEFECT TYPES OF CSS RELATED TO AFFECTED QUALITY GOAL

Mode		Sub modes		Affected Quality Goal
MISSING	= 18 defects	Missing	= 18 defects	QG2, QG4
WRONG	= 182 defects	Inconsistent	= 75 defects	QG1, QG3, QG4, QG5
		Incorrect	= 95 defects	QG1, QG4
		Ambiguous	= 12 defects	QG1, QG3,
UNNECESSARY	= 26 defects	Redundant	= 20 defects	QG5
		Extraneous	= 6 defects	QG5, QG6
TOTAL	226 defects		226 defects	

We also analysed the relation of these defect types to the quality goals proposed by Mohagheghi et al. [5]. Fig. 3 shows an overview of these results, in which Correctness (QG1) and Comprehensibility (QG4) are the quality properties with most types of identified defects in the mapping study. This could possibly be due to the fact that these quality properties are the most defect-prone at the conceptual schema level, and their correction could reduce the number of other defects such as Completeness (QG2), Consistency (QG3), Confinement (QG5) and Changeability (QG6). However, it could also be due to the support tools used in these papers mostly covering only these defect types.

Finally, since a paper could report more than one defect, we found that the paper from which most defects could be extracted was [PS1] (25 defects) and the least was [PS27] with 1 defect.

B. *RQ2: How and where have these defects been detected?*

We have identified 12 different defect detection mechanisms (DM) in the primary papers:

- M1: Analysis based on descriptive logic
- M2: Analysis of the dependency graph + reasoning procedure based in logic
- M3: Automated inspection
- M4: Checking consistency rules
- M5: Checking OCL constraints
- M6: Manual inspection of the models
- M7: Analysis on graph transformation rules
- M8: Cardinalities algorithm
- M9:Testing by model simulation
- M10:Testing executable forms
- M11:Testing of Testable Design Under Test
- M12: Black-box testing

In Table VIII the input and steps needed for each detection mechanism are summarized. From this table we can see that there is an approach [PS6] that uses two mechanisms (automated and manual inspection) for detecting defects.

In Table IX the found defect detection mechanisms are categorized according to type (Automated or Manual), tools that were used for the automated techniques and CS components.

Most of the defects reported in the CSs (82%) were detected by static techniques (i.e. where model execution is not required). From the static techniques that were used, 61% were automated by tools, while the rest were done manually.

The defect data also reveals that M6 (Manual inspection) is the mechanism with the most reported defects (62), followed by M3 (Automated inspection) (58) (see Fig. 4). On the other hand, M8 (Testing of Testable Design Under Test) found the lowest number of defects (2). These results could be due to the primary studies mostly focusing on a detection mechanism rather than reporting a complete list of defects. To further locate where the defects were found, we analysed the diagrams used to detect them.

Fig. 3. Classification of defect types based on quality goals

TABLE VIII. STEPS OF THE MECHAMISMS FOR DETECTING CS DEFECTS

DM	Prerequisites	Steps (high level)
M1	***Input***: CS ***Oracle based on***: a collection of predefined Description Logics predicates that represent structural and behavioural inconsistencies [PS26]	1. Translate the CS into DL 2. For structural inconsistencies, simply query this DL representation of the CS. 3. For behavioural inconsistencies, before querying, use the UML meta-model to obtain concrete test data to trigger behaviour.
M2	***Input***: CS with OCL constraints [PS23]. ***Oracle based on***: • the OCL constraints that come with the CS to assess external correctness, i.e. w.r.t. requirements. • a set of predefined properties for internal correctness like satisfiability, class liveness, non-redundancy, etc.	1. The CS with its external OCL constraints is translated into a logic representation. 2. Questions are formalized as derived predicates to assess the CS on these external and predefined internal properties. 3. Checking any property on a schema is reduced to checking the satisfiability of derived predicates.
M3	***Input***: a CS ***Oracle based on***: a set of heuristics that are necessary for the construction of verifiable models and the programmatic extraction of requirements gathered from practical experience (defined in [PS6])	Automated inspection of these heuristics with a dedicated tool, which internal functioning is not further described in the paper [PS6].
	Input: a CS ***Oracle based on***: a set of rules taken from the UML specification, SDMetrics[5] y particularidades de C# [PS9]]	1. Transform the CS into C# code. 2. Execute the framework for eXecutable UML (FXU) to evaluate the CS behaviour according to the rules.
	Input: a CS ***Oracle based on***: List of common defects described in [PS14]	Evaluate the CS by executing the UMLint tool that is presented in the paper [PS14] and is basically a syntactic UML checker.
	Input: a CS ***Oracle based on***: The SDMetrics[5].	Using a dedicated static UML checker tool named SAAT described in [PS16].
	Input: a CS ***Oracle based on***: • common interactive behavior, like e.g. on all traces, if a message X has been sent, then somewhere in the future on that trace message Y will be received [PS25]. • frequently occurring syntactic defects.	1. Use model checking techniques to analyze the possible interaction traces and identify patterns of common interactive behaviour. 2. During this process, detect possible ambiguities in the CS sequence diagram. 3. Apply a series of checks to identify other syntactic defects in the CS.
M4	***Input***: a CS	1. Check well-formedness of each individual CS entity using the consistency rules.
	Oracle based on: consistency rules for individual design entities and consistency rules that trace relations across various design entities. In the paper [PS5] these are defined.	2. Check relationships across entities, e.g. if an entity X requires some other entity Y, the entity Y should be defined some where in the design, else the design is inconsistent
	Input: a CS ***Oracle based on***: 24 consistency rules chosen based on the needs of industrial partners [PS11].	1. If a CS element is created, for every rule related to the element, a new instance of the rule is created and evaluated. 2. If a CS element was deleted, all intances of the element rule are destroyed. 3. For every rule instance of a changed element, an evaluation of the rules is performed.
M5	***Input***: a CS ***Oracle based on***: Predefined OCL constraints for quality of CS and/or well-formedness (syntax, basic properties, naming, best practices [PS1, PS12, PS13]	Compare the CS class diagram with the OCL constraints for detecting defects.
	Input: a CS ***Oracle based on***: OCL invariant constraints for general well-formedness of CSs[PS7]	1. For each constraint, create an ATL transformation rule. 2. Then, check the constraints by using ATL transformation
	Input: a CS with OCL constraints ***Oracle based on***: the properties that come with the CS [PS8, PS27]	Class diagrams and OCL constraints of the CS are translated into a constraint satisfaction problem and solved with different tools.
M6	***Input***: a CS ***Human oracle using***: Requirement specifications [PS2], set of heuristics based on practice experience [PS6, PS17], faults in the source code [PS15], defined collection of rules and/or metrics [PS17, PS18, PS22], list of previously identified defects [PS19]	Realize a manual inspection (or alike) on the determined artefact.
M7	***Input***: a CS ***Oracle based on***: graph transformation rules that express CS inconsistency detection and resolution [PS20].	1. Generate graphs from CS. 2. Apply the technique of critical pair analysis by using the transformation rules. 3. Analyse potential dependencies between detection and resolution of the possible inconsistencies found in the CS.
M8	***Input:*** CS class diagram with constraints [PS24]. ***Oracle based on***: • minimum cardinalities to evaluate the consistency of UML class. • cardinality is inconsistent if it can have differing	1. Transform the CS class diagram in a graph with nodes (class), edges (relationships) and a set of edge information (relation type and constraints). 2. Then, divide the graph into subgraphs in order to evaluate the relation cardinality.

[5] http://www.sdmetrics.com/

	values within the same subgraph.	
M9	***Input***: CS with associated constraints [PS4]. ***Oracle based on***: multiplicities, generalizations and OCL constraints to assess the inconsistencies between class diagram and collaboration diagrams.	1. Set the test criteria (i.e. association-end multiplicity, generalization, class attributes). 2. Create a test set in a formal semantic. 3. Determine variables values from class diagrams. 4. Determine paths from collaboration diagrams. 5. Exercise scenarios to evaluate the constraints.
M10	***Input***: CS with constraints [PS10]. ***Oracle based on***: test input constraints generates from the paths of a Variable Assignment Graph to assess the consistency between different views and constraints based on requirements.	1. Set the test adequacy criteria (i.e. all message coverage, condition coverage and all message path coverage) 2. Transform the CS in a executable form (DUT) by using Java-like Action Language format. 3. Test inputs are derived from DUT by using a Variable Assignment Graph. 4. Exercise the executable form of a DUT with test inputs.
M11	**Input**: CS with OCL constraints [PS21] . ***Oracle based on***: • OCL pre-conditions, post-conditions and invariants that come with the CS to assess inconsistencies. • a set of heuristics that are necessary for the construction of verifiable OCL expressions	1. Build a Testable Aggregate Model (TAM) by combining information from the class diagram, sequence diagram and OCL information of the CS. 2. Generate the test cases and input values. 3. Then, the TAM is evaluated by using a symbolic execution. 4. Finally, the tool USE is used for parsing and validating OCL expressions.
	(defined in USE6 tool)	
	Input: a CS activiy diagram [PS28] ***Oracle based on***: path conditions and input values are used to find the inconsistency between the design and implementation.	1. CS activiy diagram is parsed and initialized semantically with concrete data. 2. The model is symbolically executed, to collect paths, input variables, and their path conditions. 3. Then, the path conditions are passed to a constraint solver to generate a set of concrete value of possible input variables. 4. Finally, the generated concrete input variables are semantically executed on the model to identify the defects.
M12	***Input***: a CS with OCL constraints [PS3]. ***Oracle based on***: the OCL constraints that come with the CS to assess if the CS meets the user's requirements.	1. Transforms a CS class diagram into test data for the logical animation. 2. The constraints defined in a class diagram are transformed into SQL queries into a Java class called TestCases. 3. JUnit tool is used to run these test cases.

In Table IX, we can see that the class diagram (CD) is used in most primary studies (86%), which suggests that the structural part of the CS is the part most often used for detecting defects. The second most used is the sequence diagram (SD) in 46% of the primary studies, followed by the state machine (STDM) and activity diagrams (AD) that are used in 36% and 25% of the primary studies respectively (see Fig. 5). This confirms that these four diagrams are the most frequently used to specify both the structural and behavioural part of a CS Techniques used as defect detection mechanism.

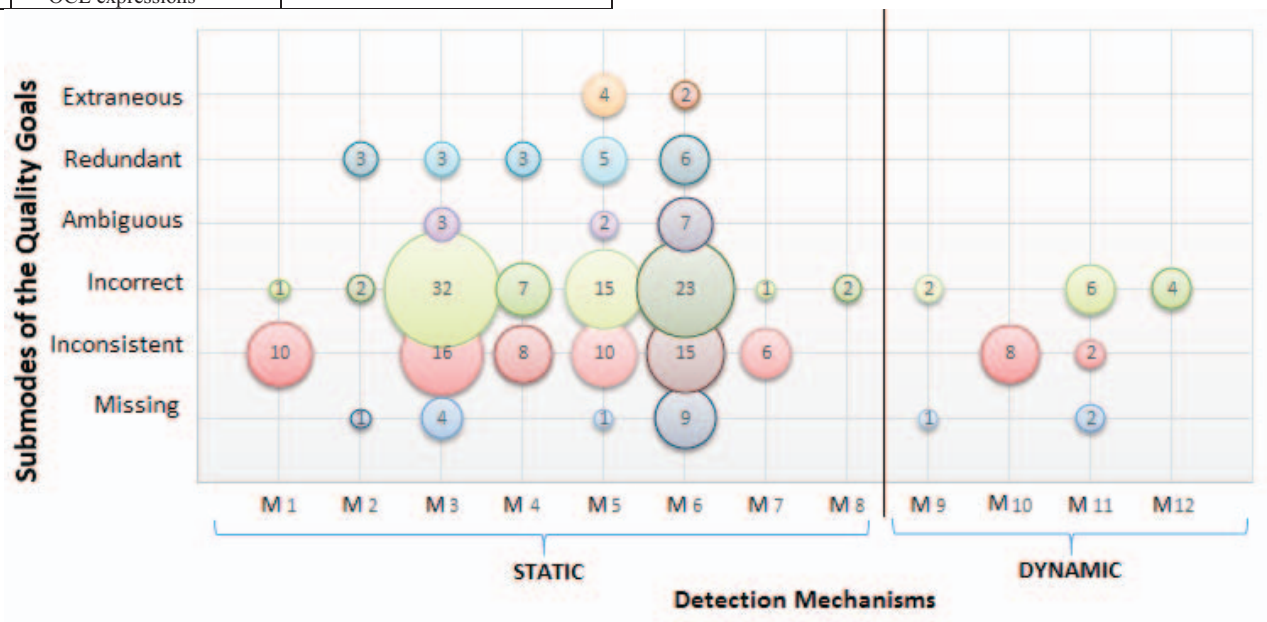

Fig. 4. Number of defects grouped by Detection Mechanism and Sub mode of the QGs

[6] http://sourceforge.net/projects/useocl/

TABLE IX. TECHNIQUES USED AS DEFECT DETECTION MECHANISM

Tech-nique	DM	Type	Tool	UML Diagrams		Ref.
				Structural	Behavioural	
Static	M1	A	RACOoN	CD	SD, CoD, STMD	PS26
	M2		EinaGMC	CD	-	PS23
	M3	M/A	Design Advisor	CD	UC, AD, CoD, SD	PS6
		Automated (A)	FXU	CD	STMD	PS9
			UMLint	CD	UC	PS14
			SDMetrics, SAAT	CD	UC, SD	PS16*
			SquAT	CD	SD	PS25
	M4		Eclipse plug-in	CD, DD CompD	UC, SD, STMD	PS5
			UML/ Analyzer	CD	SD, STMD	PS11
	M5		Eclipse-based Plug-in	CD	-	PS7
			UMLtoCSP	CD	-	PS8
			USE	CD, OD	UC, SD, CoD,STMD, AD	PS13
			Executable OCL checker	-	AD	PS27
		Manual (M)	-	CD	UC , SD, STMD	PS12
			-	CD	-	PS1
	M6		-	-	STDM	PS2
			-	-	AD	PS15
			-	CD	UC, SD, STMD	PS17
			-	CD	SD, STMD	PS18
			-	CD	-	PS19
			-	CD	SD	PS22
	M7	A	AGG	CD	STMD	PS20
	M8	M	-	CD	-	PS24
Dynamic	M9		-	CD	CoD	PS4
	M10	A	Eclipse Plug-in, USE	CD	SD, AD	PS10
	M11		USE, ADAPTUML	CD	SD	PS21
			ADSim	-	AD	PS28
	M12		JUnit	CD	-	PS3

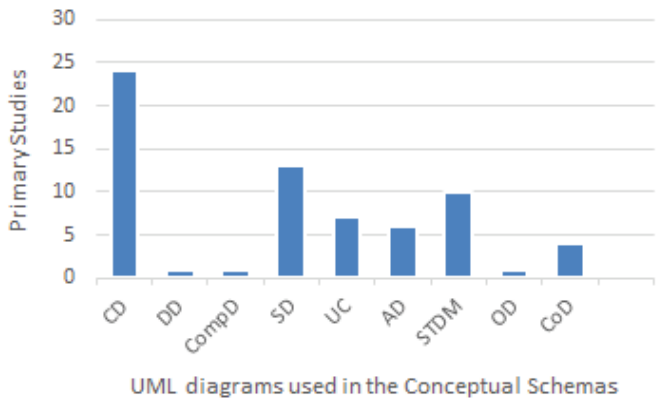

Fig. 5. Diagrams reported in the pimary studies

In order to compare the coverage of the detection mechanisms, the defect data was grouped by detection mechanism and by the causes at fine-grained level (sub modes) for CSs defects. From this data (see Fig. 4) we can see (by column) that M3, M5 and M6 to some extent tried to cover all the causes (six sub modes) of the defects reported in the papers analysed, while this is not the case for M8 and M12, which cover only one defect cause. The figure also reveals (by row) that the cause of the Incorrect sub-mode is covered by almost all mechanisms (11), followed by Inconsistent (8) and Missing (6). This is not surprising, as the main QGs pursued in such artefacts are Correctness, Completeness and Consistency [5]. Finally, whereas the primary studies analysed mainly report both correctness and consistency defects in CSs by assessing their structural properties, the dynamic mechanisms are not being fully exploited (few defects reported with these mechanisms) as testing and debugging tools in the quality assurance process of conceptual schemas.

In order to answer the second part of our research question (location of the defects) we identified the modelling element it was located in. For this we used the UML terminology [2] to distinguish 100 different elements. In Table X we can see that the modelling elements most reported in the primary studies (i.e. number of references per element) are Operation (33 references, 8%) and Class (27 references, 10%).

TABLE X. MODELLING ELEMENTS

Modelling Element	# Defects	# Refs
Action	1	1
Activity	1	1
Actor	2	2
Association	6	7
Class	**10**	**27**
Comment	1	2
Connector	3	6
Constraint	5	14
ControlNode	1	1
DecisionNode	1	2
Diagram	3	3
Feature	1	2
Generalization	**12**	21
InstanceSpecification	2	10
Interface	2	3
Lifeline	1	1
Link	1	1
Message	1	1
MessageOccurrenceSpecification	1	1
MultiplicityElement	3	7
NamedElement	3	19
Operation	**8**	**33**
Package	1	1
Property	**11**	16
ProtocolTransition	1	3
Pseudostate	1	1
Region	1	2
RelationShip	1	5
State	4	8
Transition	3	10
UseCase	2	5
ValueSpecification	2	2
VisibilityKind	4	8
Total	100	226

The study also found that Generalization is the most affected modelling element (12 defects, or 12%) by defects reported, followed by the Property and Class elements with 11 and 10 defects, respectively. This is possibly because these are the modelling elements that most occur in a structural diagram.

VI. LIMITATIONS OF THE STUDY

In the process of conducting our mapping study, we faced various limitations related to the selection of relevant studies and accuracy of the data extraction, such as the following:

Other terms such as "bug", and "error" were not considered because they are most used at code level. This was confirmed by including the terms in the search string and confirming that the papers contained no defects.

Since primary studies are mostly focused on presenting a detection mechanism rather than reporting a complete list of the defects found by this mechanism, the defect list most frequently reported in the literature is not entirely equivalent to the number of defects found in CSs. Another limitation could arise from our categorizations and its completeness for analysis, a validity threat common to mapping studies. We follow a methodology proposed in the literature [5] to create the classification scheme and the scheme attributes are based mainly on the IEEE standard [8].

Regarding the accuracy of the data extraction; several articles did not given sufficient information regarding the attributes considered for describing defects (e.g. severity and priority).

VII. CONCLUSIONS AND FUTURE WORK

This paper describes a systematic mapping study of the defect types reported on in CSs represented with UML and the techniques used to detect them. The results are summarized below.

RQ1: What defects in UML-based CSs are reported in the literature? We identified the defect types relevant to CSs by a defect classification scheme. This classification is clearly differentiated from others (e.g. defects at level code) by not considering problems related for instance to user interface and input/output crash.

Although several categories can be obtained (e.g. by severity, by defect cause, by priority), in this study our analysis was mainly carried out based on defect causes that affected quality goals defined for CSs in an MDD context [4]. We found there is a tendency to report only defect types related to the "Wrong" type (e.g. incorrect) rather than the "Missing" or "Unnecessary" types.

On the other hand, few papers reported other important aspects of the defects, such as severity (i.e. [PS7], [PS9]). No paper reported the priority of defects, which could be due to several reasons. It is possible that the complexity of managing the impact of defects on the quality of the different CS makes it difficult to specify values for these attributes, i.e. the lack of a history of defects makes it difficult to establish their severity and priority. These findings indicate that more research is needed to better understand the severity and priority of defects with respect to their impact on the quality properties.

RQ2: How and where have these defects been detected? We identified some of the techniques used for detecting defects. However, the evidence from this review suggests that the main emphasis is on the use of techniques based on static analysis of the CS, which does not require the execution of the model, so that only part of the specification can be analysed.

Performing a systematic mapping study is time-consuming and interpreting the results is not easy. However, the main benefits are that it provides conclusions, new insights and identifies research gaps.

After analysing the results of the review, we can draw the conclusion that although there are several studies aimed at reporting defect types at the CS level, a complete, well-documented (e.g. based on defect classification scheme) and evaluated list is still lacking. This information should include an analysis of the severity of the defects with respect to their impact on the quality properties of CS and the priority in solving them.

To date, defect types appear to be poorly used since most of them are used primarily to verify the *Correctness* and *Comprehensibility* of a CS by static detection techniques. Therefore, more efforts (e.g. by using dynamic techniques such as testing) are required to detect other defect types at the conceptual schema level and to exploit the information contained in them e.g. to measure the efficiency and effectiveness of verification[7] & validation[8] tools with respect to the number and type of defects that they manage. Additionally, with this information the defects detected in an MDD environment could be tracked, reduced and resolved.

One of the challenges involved in this review was developing an appropriate defect classification scheme for CSs. Although we have gone through a couple of revisions throughout the study, we found that our final classification scheme is highly usable and complete for our mapping.

This systematic mapping study is a part of a more extensive research work, whose principal goal is to propose an approach for testing-based conceptual schema validation in a Model-Driven Environment [18], [19]. To reach that goal, further research is needed. First, we need to clarify which defect types can be found with testing techniques. Second, we need to know which parts of a CS are expected to be most defect-prone. Third, we need to prioritize defects types that are expected to appear most often during testing activities. Finally, we need to develop a testing solution and evaluate it by and deploying it in an MDD environment.

ACKNOWLEDGMENT

This work has been supported by the Secretary of Higher Education, Science and Technology (SENESCYT: Secretaría Nacional de Educación Superior, Ciencia y Tecnología), of the Republic of Ecuador.

[7] Verification is to check that the CS meets its stated functional and non-functional requirements [26].

[8] Validation is to ensure that the CS meets the customer's expectations [26].

APPENDIX I

TABLE XI. COMPLIANCE OF OUR PROPOSED CLASSIFICATION SCHEME WITH IEEE STD. 1044

IEEE Std. 1044	Definition	Our Proposal
Defect ID	Unique identifier for the failure.	Defect ID
Description	Description of what is missing, wrong, or unnecessary.	Description
Status	Current state within defect report life cycle.	n/a
Asset	The software asset (product, component, module, etc.) containing the defect.	Diagram Type
Artefact	The specific software work product containing the defect.	Modelling Element
Version Detected	Identification of the software version in which the defect was detected.	References
Version Corrected	Identification of the software version in which the defect was corrected.	n/a
Priority	Ranking for processing assigned by the organization responsible for the evaluation, resolution, and closure of the defect relative to other reported defects.	Priority
Severity	The highest failure impact that the defect could (or did) cause, as determined by (from the perspective of) the organization responsible for software engineering.	Severity
Probability	Probability of recurring failure caused by this defect.	n/a
Effect	The class of requirement that is impacted by a failure caused by a defect.	n/a
Type	A categorization based on the class of code within which the defect is found or the work product within which the defect is found.	Diagram level
Mode	A categorization based on whether the defect is due to incorrect implementation or representation, the addition of something that is not needed, or an omission.	Defect Cause
Insertion Activity	The activity during which the defect was injected/inserted (i.e., during which the artifact containing the defect originated).	n/a
Detection Activity	The activity during which the defect was detected (i.e., inspection or testing).	Detection Mechanism
Failure Reference(s)	Identifier of the failure(s) caused by the defect.	n/a
Change Reference	Identifier of the corrective change request initiated to correct the defect.	n/a
Disposition	Final disposition of defect report upon closure.	n/a
-		Technique Purpose
-		Technique Type
-		Tool Support

APPENDIX II

TABLE XII. LIST OF PRIMARY STUDIES INCLUDED IN THE REVIEW

Ref	Source
[PS1]	Aguilera, D., Gómez, C., Olivé, A., Enforcement of Conceptual Schema Quality Issues in Current Integrated Development Environments. In CAiSE 2013, Valencia, Spain (2013).
[PS2]	Ali, S., Yue, T., Malik, Z.: Comprehensively evaluating conformance error rates of applying aspect state machines. In : Proceedings of the 11th annual international conference on Aspect-oriented Software Development, pp.155-166 (2012)
[PS3]	Aljumaily, H., Cuadra, D., Martinez, P.: Applying black-box testing to UML/OCL database models. Software Quality Journal 22(2), 153-184 (2014)
[PS4]	Andrews, A., France, R., Ghosh, S., Craig, G.: Test adequacy criteria for UML. Software Testing Verification and Reliability 13(2), 95-127 (2003)
[PS5]	Bellur, U., Vallieswaran, V.: On OO design consistency in iterative development. In IEEE, ed. : in ITNG '06, pp.46-51 (2006)
[PS6]	Berenbach, B.: The evaluation of large, complex UML analysis and design models. In Society, I., ed. : in Proceedings ICSE '04, pp.232-241 (2004)
[PS7]	Bezivin, J., Jouault, F.: Using ATL for checking models. Electronic Notes in Theoretical Computer Science 1(2), 70-118 (2006)
[PS8]	Cabot, J., Clarisó, R., Riera, D.: Verification of UML/OCL Class Diagrams using Constraint Programming. In Proceedings ICSTW '08, 73-80 (2008)
[PS9]	Derezinska, A., Pilitowski, R.: Correctness issues of UML Class and State Machine Models in the C# Code Generation and Execution Framework. In Proceedings IMCSIT'2008.
[PS10]	Dinh-Trong, T., Ghosh, S., France, B.: A Systematic Approach to Generate Inputs to Test UML Design Models. In Proceedings ISSRE, 95-104 (2006)
[PS11]	Egyed, A.: Instant consistency checking for the UML. In Proceedings ICSE '06, Shanghai, China, pp.381-390 (2006)
[PS12]	Gomaa, H., Wijesekera, D. : Consistency in multiple-view UML models: a case study. In IEEE, ed. : in Proceedings of the Workshop on Consistency Problems in UML-based Software Development II, San Francisco, Calif, USA, pp.1-8 (2003)
[PS13]	Ha, I., Kang, B.: Meta-Validation of UML Structural Diagrams and Behavioral Diagrams with Consistency Rules. Communications, Computers and Signal Processing 2, 679-683 (2003)
[PS14]	Hasker, R. W., Rowe, M., UMLint: Identifying defects in UML diagrams, *in ASEE' 2011*, Vancouver, BC, Canada, 2011.
[PS15]	Holt, N. E., Briand, L. C., Torkar, R.: Empirical evaluations on the cost-effectiveness of state-based testing: An industrial case study. Inf. and Software Technology 56(8), 890-910 (2014)
[PS16]	Lange, C., Chaudron, M. : Defects in industrial UML models—a multiple case study. In Proceedings QiM '07, Nashville, Tenn, USA, pp.50-79 (2007)
[PS17]	Lange, C., Chaudron, M.: An empirical assessment of completeness in UML designs. In IEEE, ed. : in Proceedings EASE '04, pp.111-121 (2004)
[PS18]	Lange, C., Chaudron, M., Muskens, J., Somers, L., Dortmans, H.: An empirical investigation in quantifying inconsistency and incompleteness of uml designs. In : Proc. Workshop on Consistency Problems in UML-based Software Development, 6th International Conference on Unified Modelling Language, UML 2003, San Francisco, USA (2003)
[PS19]	Leung, F., Bolloju, N.: Analyzing the Quality of Domain Models Developed by Novice Systems Analysts. In Proceedings HICSS'2005, Hawaii, pp.1-7 (2005)
[PS20]	Mens, T., Van Der Straeten, R., D'Hondt, M.: Detecting and resolving model inconsistencies using transformation dependency analysis. In MoDELS 2006 4199 of LNCS, 200–214 (2006)
[PS21]	Pilskalns, O., Andrews, A., Knight, A., Ghosh, S., France, R.: Testing UML designs. In : Information and Software Technology, MA, USA (2007)
[PS22]	Pilskalns, O., Williams, D., Aracic, D., Andrews, A.: Security consistency in UML designs. In : COMPSAC '06. 30th Annual International Computer Software and Applications Conference, Chicago, IL, pp.351 - 358 (2006)
[PS23]	Queralt, A., Teniente, E.: Verification and validation of UML conceptual schemas with OCL. ACM Transactions on Software Engineering and Methodology (2012)
[PS24]	Satish, S. S., Shashikant, S. R., Sambhe, V. K., Shelke, R. B., Kocharekar, G.: A minimum cardinality consistency-checking algorithm for UML class diagrams. In : Proc. of the International

	Conference and Workshop on Emerging Trends in Technology, pp.222-223 (2010)
[PS25]	Van Amstel, M. F., Lange, C. F. J., Chaudron, M. R. V.: Four Automated Approaches to Analyze the Quality of UML Sequence Diagrams. In Proc. COMPSAC 2007, pp.415-424 (2007)
[PS26]	Van Der Straeten, R.: Inconsistency Management in Model-Driven Engineering. An Approach using Description Logics PHD Thesis ed. Faculty of Science, Brussels (2005)
[PS27]	Yin, L., Liu, J., Li, X.: Validating requirements model of a B2B system. In : ICIS 2009. Eighth IEEE/ACIS International Conference on Computer and Information Science, Shanghai, pp.1020 - 1025 (2009)
[PS28]	Yu, L., X., T., L., W., X., L.: Simulating software behavior based on UML activity diagram. In : Proceedings of the 5th Asia-Pacific Symposium on Internetware (2013)

APPENDIX III

TABLE XIII. PARTIAL OVERVIEW OF THE DEFECTS LIST REPORTED IN THE MAPPING STUDY

Mode	Defect ID	Description	Modelling Element	Tech-nique Type	Ref.
Missing	D1.1	Missing Class specification	Class	Static	PS16
Wrong	D2.1	Inconsistent Operation access permissions	Visibility Kind	Static and Dyna mic	PS21 PS22
	D3.21	Incorrect Multiplicity definition	Multiplicity Element	Static	PS14 PS19 PS24
	D4.2	Ambiguous Discriminator Constraint	Generalization	Static	PS19
Unnecessary	D5.11	Redundant Constraint	Constraint	Static	PS8 PS23
	D6.3	Extraneous Too many attributes	Property	Static	PS1

REFERENCES

[1] Y. Labiche, "The UML Is More Than Boxes and Lines," in MODELS, 2008, pp. 375-386.

[2] Object Management Group. Unified Modeling Language (UML) Versión 2.5.

[3] OMG. (2006) Object Constraint Language OMG Available Specification version 2.0 formal/06-05-01. [Online]. http://www.omg.org/spec/OCL/2.0/PDF/

[4] B. Unhelkar, Verification and Validation for Quality of UML 2.0 Models. New Jersey: John Wiley & Sons, Inc., 2005.

[5] P. Mohagheghi, V. Dehlen, and T. Neple, "Definitions and approaches to model quality in model-based software development– A review of literature," Information and Software Technology, vol. 51, pp. 1646-1669, 2009.

[6] B. Freimut, "Developing and Using Defect Classification Schemes," Institut Experimentelles Software Engineering, Sauerwiesen, IESE-Report No.072.01/E, 2001.

[7] R. B Grady, "Software Failure Analysis for High-Return Process Improvement Decisions," Hewlett-Packard Journal, 1996.

[8] R. Chillarege, "Orthogonal Defect Classification," in Handbook of Software Reliability Engineering, IEEE Computer Society Press and McGraw-Hill ed.: M. R. Lyu, 1996.

[9] IEEE. (2010) IEEE Std. 1044-2009. Standard Classification for Software Anomalies.

[10] S. Wagner, "Defect Classification and Defect Types Revisited," in Proceedings of the 2008 workshop on Defects in large software systems, NY, USA, 2008, pp. 39–40.

[11] R. Conradi et al., "Object-oriented reading techniques for inspection of UML models – an industrial experiment," in Proceedings of ECOOP'03, vol. 2749 of LNCS, 2003.

[12] G. H. Travassos, F. Shull, J. Carver, and V. Basili, "Reading Techniques for OO Design Inspections," UM Computer Science Department, Technical Report CS-TR-4353 UMIACS; UMIACS-TR-2002-33, 2002.

[13] T. Dinh-Trong, S. Ghosh, R. France, B. Benoit, and F. Fleury, "A taxonomy of faults for uml designs," in MoDELS'05, Montego Bay, Jamaica, 2005.

[14] J. P. Simmonds, "Consistency Maintenance of UML Models with Description Logics," Vrije Universiteit Brussel, Brussel, Thesis of Master of Science in Computer Science 2003.

[15] A. Tort and A. Olivé, "An approach to testing conceptual schemas," , vol. 69, 2010, pp. 598-618.

[16] B. A. Kitchenham, D. Budgen, and O. P. Brereton, "Using mapping studies as the basis for further research – A participant-observer," vol. 53, pp. 638-651, July 2011.

[17] K. Petersen, R. Feldt, S. Mujtaba, and M., Mattsson, "Systematic Mapping Studies in Software Engineering," in Proceedings EASE 08, BCS eWIC, 2008.

[18] M. F. Granda, N. Condori-Fernandez, T. E. J. Vos, and O. Pastor, "Towards the Automated Generation of Abstract Test Cases from Requirements Models ," in Requirements Engineering and Testing (RET), 2014 IEEE 1st International Workshop on, Karlskrona, Sweden, 2014, pp. 39-46.

[19] M. F. Granda, "Testing-Based Conceptual Schema Validation in a Model-Driven Environment," in CAiSE'2013, Doctoral Consortium, vol. 1001, Valencia, 2013. [Online]. http://ceur-ws.org/Vol-1001/

[20] Sindre G., Sølvberg A. Lindland O. I., "Understanding Quality in Conceptual Modeling," IEEE Software, vol. 11, no. 2, pp. 42-49, 1994.

[21] J. Krogstie, Model-Based Development and Evolution of Information Systems. A Quality Approach. London: Springer, 2012.

[22] A. Jalbani, M. Memon, I. Qureshi, and A. Yasmin, "A Novel Quality Model for UML Models," in ICCIT 2012, 2012, pp. 248-252.

[23] F. Shull, G. H. Travassos, and V. Basili, "Towards Techniques for Improved OO Design Inspections," in Workshop on Quantitative Approaches in Object-Oriented Software Engineering, Lisbon, Protugal, 1999.

[24] IEEE. (1996) IEEE Std. 1044.1-1995 - Guide to Classification for Software Anomalies.

[25] A. Olivé, Conceptual Modeling of Information System. Spain: Springer, 2007.

[26] I. Sommerville, Software Engineering, 9th ed. Boston, Mass.: Addison-Wesley, 2011.